

An Additive Model View to Sparse Gaussian Process Classifier Design

Sundararajan S¹ and Shirish Shevade²

¹ Yahoo! Labs, Bangalore, India,
ssrajan@yahoo-inc.com

² Department of Computer Science and Automation
 Indian Institute of Science
 Bangalore, India
shirish@csa.iisc.ernet.in

Abstract. We consider the problem of designing a sparse Gaussian process classifier (SGPC) that generalizes well. Viewing SGPC design as constructing an additive model like in boosting, we present an efficient and effective SGPC design method to perform a stage-wise optimization of a predictive loss function. We introduce new methods for two key components viz., site parameter estimation and basis vector selection in any SGPC design. The proposed adaptive sampling based basis vector selection method aids in achieving improved generalization performance at a reduced computational cost. This method can also be used in conjunction with any other site parameter estimation methods. It has similar computational and storage complexities as the well-known information vector machine and is suitable for large datasets. The hyperparameters can be determined by optimizing a predictive loss function. The experimental results show better generalization performance of the proposed basis vector selection method on several benchmark datasets, particularly for relatively smaller basis vector set sizes or on difficult datasets.

Key words: Gaussian process, Classification, Sparse models, Additive models

1 Introduction

Sparse Gaussian Process (GP) classifier design aims at addressing the issues of high computational and storage costs associated with learning a full model GP ($O(n^3)$ and $O(n^2)$ respectively) [6] using n training examples, and involves using a representative data set, called the *basis vector* set, from the input space. In this way, the computational and memory requirements are reduced to $O(nd_{max}^2)$ and $O(nd_{max})$ respectively, where d_{max} is the size of the basis vector set ($d_{max} \ll n$). Further, the costs of predictive mean and variance computations for an example are reduced from $O(n)$ and $O(n^2)$ to $O(d_{max})$ and $O(d_{max}^2)$ respectively.

In this work, we focus on developing an efficient Sparse Gaussian Process Classifier (SGPC) design algorithm. Several approaches have been proposed in the literature to design sparse GP classifiers. These include on-line GP learning [1] and entropy or information gain based Informative Vector Machine (IVM) [4,8]. Particularly relevant to this work is IVM which is inspired by the technique of assumed density filtering (ADF) [5,1]. In general, an SGPC design algorithm using the ADF approximation involves site parameter estimation, basis vector selection and hyperparameter optimization. While the site parameters are estimated using a moment matching technique in the ADF approximation, hyperparameters are estimated by optimizing marginal likelihood or negative logarithm of predictive probability (NLP) [6]. Different methods to select the basis vectors include entropy, information gain and validation based methods [9]. Experimental comparisons of the IVM with entropy based method and validation based method on various benchmark datasets showed that though the IVM method is efficient, it does not generalize well particularly on difficult datasets, and it requires more number of basis vectors to achieve similar generalization performance compared to the validation based method. Though the validation based method generalizes well, it is computationally expensive. Therefore, there is a need to have an efficient algorithm to design SGPCs that generalize well.

Contributions: Viewing SGPC design as construction of an additive model (that is, a linear combination of basis functions) [3], a basis vector addition can be seen as adding a basis function in each iteration like in boosting [7]. With this view we introduce new methods to select the basis vectors and, estimate their site parameters by optimizing a predictive loss function. These estimated site parameters determine the coefficient of the basis function in the additive model. Further, an adaptive sampling based basis vector selection method is proposed, which aids in effective basis vector selection and computational cost reduction. The proposed basis vector selection method has same computational complexity as used by IVM. We also compare the generalization performance of various basis vector selection methods. Experimental results show that the proposed method gives comparable or better performance on a wide range of real-world large datasets. In particular, the proposed method is significantly better compared to the entropy and information gain based methods for relatively smaller d_{max} values or on difficult datasets.

The paper is organized as follows. Section 2 presents an SGPC design algorithm with the ADF approximation. The proposed methods and implementation

aspects are given in Section 3. Section 4 covers related work. Experimental results are presented in Section 5 and the paper concludes with Section 6.

2 GP and Sparse GP Classification

Given a training data set with input-output pairs $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^n$ where $\mathbf{x}_i \in R^d$ and $y_i \in \{+1, -1\}$, the goal is to design a GP classifier that generalizes well. In standard GPs for classification [6], true function value at each \mathbf{x}_i is represented as a latent random variable $f(\mathbf{x}_i)$. Let us denote $f(\mathbf{x}_i)$ by f_i . The prior distribution of $\{f(\mathbf{X}_n)\}$ is a zero mean multivariate joint Gaussian, denoted as $p(\mathbf{f}) = \mathcal{N}(\cdot; \mathbf{0}, \mathbf{K})$, where $\mathbf{f} = [f_1, \dots, f_n]^T$, $\mathbf{X}_n = [\mathbf{x}_1, \dots, \mathbf{x}_n]$, and \mathbf{K} is an $n \times n$ covariance matrix whose $(i, j)^{th}$ element is $k(\mathbf{x}_i, \mathbf{x}_j)$. An example covariance function is the squared exponential function: $k(\mathbf{x}_i, \mathbf{x}_j) = v_0 \exp(-\frac{1}{2} \sum_{m=1}^d \frac{(x_{i,m} - x_{j,m})^2}{\sigma^2})$. Here, v_0 and σ^2 denote the signal variance and kernel width respectively. In this work we use the probit noise model, $p(y_i | f_i, \lambda, b) = \Phi(\lambda y_i (f_i + b))$ where $\Phi(\cdot)$ is the cumulative distribution of the standard Gaussian $\mathcal{N}(\cdot; 0, 1)$ with zero mean and unit variance, the slope of which is controlled by $\lambda(>0)$ and b is a bias hyperparameter. With independent, identical distribution assumption, we have $p(\mathbf{y} | \mathbf{f}, \gamma) = \prod_{i=1}^n p(y_i | f_i; \gamma)$ where $\gamma = [\lambda, b]$. Let $\boldsymbol{\theta} = [v_0, \sigma^2, \gamma]$ denote the hyperparameters that characterize the GP model. With these modeling assumptions, the expressions for latent posterior and predictive distributions are available [6]. In SGPC design using the ADF approximation [4], a factorized form of $q_{\mathbf{u}}(\mathbf{f} | \mathcal{D}, \boldsymbol{\theta})$ (given below) is made use of, to build an approximation to $p(\mathbf{f} | \mathcal{D}, \boldsymbol{\theta})$ in an incremental fashion. Let \mathbf{u} denote the index set of the training examples which are included in the approximation. Then we have

$$q_{\mathbf{u}}(\mathbf{f} | \mathcal{D}, \boldsymbol{\theta}) \propto \mathcal{N}(\mathbf{f}; \mathbf{0}, \mathbf{K}) \prod_{i \in \mathbf{u}} \exp \left\{ -\frac{p_i}{2} (f_i - m_i)^2 \right\} \quad (1)$$

and $p(\mathbf{f} | \mathcal{D}, \boldsymbol{\theta}) \approx q_{\mathbf{u}}(\mathbf{f} | \mathcal{D}, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{f}; \hat{\mathbf{f}}, \mathbf{A})$ where $\mathbf{A} = (\mathbf{K}^{-1} + \boldsymbol{\Pi})^{-1}$ and $\hat{\mathbf{f}} = \mathbf{A} \boldsymbol{\Pi} \mathbf{m}$, $\mathbf{m} = (m_1, \dots, m_n)^T$ and $\boldsymbol{\Pi} = \text{diag}(p_1, \dots, p_n)$. The parameters m_i and p_i , $i = 1 \rightarrow n$ are called the site function parameters and the set \mathbf{u} is called the *active* or *basis vector* set. Note that \mathbf{u} is actually associated with the inputs $\mathbf{X}_{\mathbf{u}}$. We refer to $\mathbf{u}^c = \{1, 2, \dots, n\} \setminus \mathbf{u}$ as the non-active set. In practice, the active set size $|\mathbf{u}|$ is restricted by the user specified parameter, d_{max} . Note that the site function parameters corresponding to \mathbf{u}^c are zero. Thus a SGPC model is defined by the basis vector set \mathbf{u} , its associated site function parameters $(\mathbf{m}_{\mathbf{u}}, \boldsymbol{\Pi}_{\mathbf{u}})$ and the hyperparameters $\boldsymbol{\theta}$. In general, SGPC design algorithms differ with respect to the basis vector selection, site parameter estimation and hyperparameters optimization methods. A typical SGPC design algorithm using the ADF approximation is given in Algorithm 1.

We now briefly describe the ADF approximation method [4] to implement step 4. Suppose that an example index j is added to the current basis vector set \mathbf{u} . Let $\bar{\mathbf{u}}_j = \mathbf{u} \cup \{j\}$. After updating the site function parameters p_j and m_j , incremental calculations are carried out to update $\hat{\mathbf{f}}$ and $\text{diag}(\mathbf{A})$ corresponding to

Algorithm 1 SGPC Design

-
1. Initialize the hyperparameters θ . Set d_{max} , tol , $iter_{max}$ and, $iter=0$.
 - repeat**
 2. Initialize $\mathbf{A} := \mathbf{K}$, $\mathbf{u} = \{\}$, $\mathbf{u}^c = \{1, 2, \dots, n\}$, $\hat{f}_i = p_i = m_i = 0 \ \forall i \in \mathbf{u}^c$.
 $iter = iter + 1$.
 - repeat**
 3. Select a basis vector j from \mathbf{u}^c as per the chosen basis vector selection method.
 4. Update the site parameters p_j , m_j , posterior mean ($\hat{\mathbf{f}}$) and variance ($\text{diag}(\mathbf{A})$).
 5. Set $\mathbf{u} = \mathbf{u} \cup \{j\}$ and $\mathbf{u}^c = \mathbf{u}^c \setminus \{j\}$.
 - until** $|\mathbf{u}| = d_{max}$
 6. Re-estimate the hyperparameters θ by optimizing a suitable loss function, keeping \mathbf{u} and the corresponding site parameters constant.
 - until** $iter = iter_{max}$ or change in the loss function value $< tol$
-

$\bar{\mathbf{u}}_j$. This is achieved by maintaining two matrices \mathbf{L} and \mathbf{M} where \mathbf{L} is the lower-triangular Cholesky factor of $\mathbf{B} = \mathbf{I} + \boldsymbol{\Pi}_{\mathbf{u}, \mathbf{u}}^{1/2} \mathbf{K}_{\mathbf{u}, \mathbf{u}} \boldsymbol{\Pi}_{\mathbf{u}, \mathbf{u}}^{1/2}$ and $\mathbf{M} = \mathbf{L}^{-1} \boldsymbol{\Pi}_{\mathbf{u}, \mathbf{u}} \mathbf{K}_{\mathbf{u}, \cdot}$.³ Note that $\mathbf{A} = \mathbf{K} - \mathbf{M}^T \mathbf{M}$. However, only the diagonal elements of \mathbf{A} are needed in the algorithm and are updated as given in (3) below. Assuming $\lambda = 1$, with $z_j = \frac{y_j(\hat{f}_j + b)}{\sqrt{1 + A_{jj}}}$, $\alpha_j = \frac{y_j \mathcal{N}(z_j; 0, 1)}{\Phi(z_j)} \sqrt{\frac{1}{1 + A_{jj}}}$, $\nu_j = \alpha_j \left(\alpha_j + \frac{(\hat{f}_j + b)}{1 + A_{jj}} \right)$ the site function parameters are updated as:

$$p_j = \frac{\nu_j}{1 - A_{jj}\nu_j}, \quad m_j = \hat{f}_j + \frac{\alpha_j}{\nu_j}. \quad (2)$$

Let $\mathbf{l} = \sqrt{p_j} \mathbf{M}_{\cdot, j}$, $l = \sqrt{1 + p_j \mathbf{K}_{j, j} - \mathbf{l}^T \mathbf{l}}$, $\boldsymbol{\mu} = l^{-1} (\sqrt{p_j} \mathbf{K}_{\cdot, j} - \mathbf{M}^T \mathbf{l})$. Then \mathbf{M} is updated by appending the row vector $\boldsymbol{\mu}^T$ and \mathbf{L} is updated by appending $[\mathbf{L} \mathbf{0}]$ with $[\mathbf{l}^T l]$. The posterior variance and mean are updated as:

$$\text{diag}(\mathbf{A}) := \text{diag}(\mathbf{A}) - \boldsymbol{\mu}^2, \quad \hat{\mathbf{f}} := \hat{\mathbf{f}} + \alpha_j l p_j^{-1/2} \boldsymbol{\mu}. \quad (3)$$

In (3), $\boldsymbol{\mu}^2$ denotes squaring of each element in $\boldsymbol{\mu}$. In the outer loop the hyperparameters are optimized by maximizing the marginal likelihood (ML) [4], $q_{\mathbf{u}}(\mathbf{y}|\mathbf{X}, \theta) = \int p(\mathbf{y}|\mathbf{f}, \gamma) q_{\mathbf{u}}(\mathbf{f}|\mathcal{D}, \theta) d\mathbf{f}$ or minimizing the negative logarithm of predictive probability (NLP) loss (under cumulative Gaussian noise model) [9],

$$\text{NLP}(\mathbf{u}, \theta) = -\frac{1}{|\mathbf{u}^c|} \sum_{i \in \mathbf{u}^c} \log \Phi \left(\frac{y_i(\hat{f}_i + b)}{\sqrt{1 + \mathbf{A}_{ii}}} \right). \quad (4)$$

Finally the predictive target distribution for an unseen input x_* is given by:

$$q_{\mathbf{u}}(y_*|\mathbf{x}_*) = \Phi \left(\frac{y_*(\hat{f}_* + b)}{\sqrt{1 + \sigma_*^2}} \right) \text{ where } \hat{f}_* = \mathbf{k}_{*, \mathbf{u}} \boldsymbol{\Pi}_{\mathbf{u}}^{\frac{1}{2}} \mathbf{B}^{-1} \boldsymbol{\Pi}_{\mathbf{u}}^{\frac{1}{2}} \mathbf{m}_{\mathbf{u}} \text{ and } \sigma_*^2 = k(\mathbf{x}_*, \mathbf{x}_*) -$$

³ The subscript, (\mathbf{u}, \mathbf{u}) , of a matrix is used to represent the rows and columns of the matrix corresponding to the elements of the set \mathbf{u} . The subscript, (\mathbf{u}, \cdot) denotes the rows of the matrix corresponding to the elements of the set \mathbf{u} .

$\mathbf{k}_{*,\mathbf{u}} \Pi_{\mathbf{u}}^{\frac{1}{2}} \mathbf{B}^{-1} \Pi_{\mathbf{u}}^{\frac{1}{2}} \mathbf{k}_{\mathbf{u},*}$. In the next section we propose new methods for effective basis vector selection and site parameters optimization (steps 3 and 4 in Algorithm 1).

3 Proposed Methods

Friedman et al [3] showed how boosting [7] can be seen as a way of fitting an additive model, $f_M(\mathbf{x}) = \sum_{m=1}^M w_m \psi(\mathbf{x}; \delta_m)$ where w_m , $m = 1, 2, \dots, M$ are the expansion coefficients, and $\psi(\mathbf{x}; \delta_m) \in \mathcal{R}$ are the basis functions characterized by the parameters δ_m , $m = 1, 2, \dots, M$, and M is the number of basis functions ($M = d_{max}$). This model is fit by minimizing a loss function averaged over the training data, that is: $\min_{\{w_m, \delta_m\}_1^M} \sum_{i=1}^n \exp(-y_i f_M(\mathbf{x}_i))$ where an exponential loss function is used. In forward stagewise additive modeling the basis functions are added one at a time and, the coefficient and the basis function parameter (w_m, δ_m) are optimized by keeping the coefficients and parameters of the previously chosen basis functions constant. That is, $\{w_m, \delta_m\} = \arg \min_{w, \delta} \exp(-y_i f_{m-1}(\mathbf{x}_i) + w \psi(\mathbf{x}; \delta))$, $m > 1$. Friedman et al [3] also presented a related loss function that is based on the binomial likelihood, given by: $\sum_{i=1}^n \log(1 + \exp(-2y_i f_M(\mathbf{x}_i)))$.

With this view, we consider the SGPC design as constructing a forward stagewise additive model. Before we show the equivalences between the selection of a basis function and its coefficient to the selection of a basis vector (j) and its site parameters (p_j, m_j), we define an objective function (called predictive loss function) that we propose to use to select a basis function and its coefficient in each iteration of the SGPC design algorithm:

$$\text{NLP}_a(\{\mathbf{u} \cup j\}, \boldsymbol{\theta}) = -\frac{1}{n} \sum_{i=1}^n \log \Phi \left(\frac{y_i(\hat{f}_i + b)}{\sqrt{1 + \mathbf{A}_{ii}}} \right) \quad (5)$$

where $j \in \mathbf{u}^c$ and, $\hat{f}_i, \mathbf{A}_{ii}$ are computed using $\{\mathbf{u} \cup j\}$. This objective function has a behavior similar to the exponential and binomial likelihood loss functions mentioned above and, is also an upper bound on the training set error. That is, we have:

$$\frac{1}{n} |\{i : \text{sgn}(\hat{f}(\mathbf{x}_i) + b) \neq y_i\}| \leq \frac{1}{\log(2)} \text{NLP}_a(\mathbf{u}, \boldsymbol{\theta}) \quad (6)$$

Here the left hand side represents the training set error. The inequality follows from noting that $0 \leq \Phi(z) \leq 1$, $\Phi(0) = 0.5$ and that $-\log(\Phi(z))$ monotonically decreases in the interval $(-\infty, \infty)$. Note that $\log(\Phi(0)) = -\log(2)$ and is required for appropriate scaling so that $-\frac{\log(\Phi(z))}{\log(2)} \geq 1$ when $z \leq 0$. Thus (5) is an upper bound on the training set error.

Comparison of Objective Functions: Firstly, unlike the exponential loss function used in boosting, the function $\Phi(\cdot)$ is not separable. That is, the linear combination of basis functions that appear inside $\Phi(\cdot)$ cannot be written as a product of individual terms. This separability property of the exponential function is useful for the interpretation of building successive weak classifiers on the

training data with weighted distribution. However, keeping all the previous basis functions with the associated coefficients fixed and, optimizing over *only* an additional basis function along with its coefficient using (5) essentially has the same desirable effect. It may be noted that like (5), the binomial log likelihood is not separable in strict sense (without any approximation). Secondly, the GP classifier has the advantage of providing predictive variance information which is useful in moderating the predictive probability. Specifically when the uncertainty or variance is large, this probability gets reduced accordingly. This is very important particularly when the data points are sparse in a certain region of the input space or when the data is noisy. Thus, use of (5) would be more robust. The behaviors of $-\log(\Phi(\cdot))$ with and without moderation along with the other loss functions are shown in Figure 1.

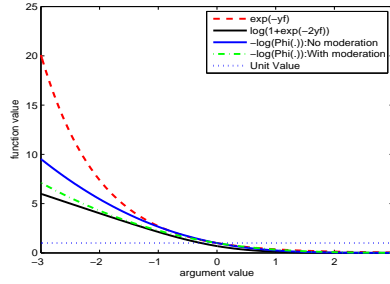


Fig. 1. Exponential, binomial log likelihood, $-\frac{\log(\Phi(\cdot))}{\log(2)}$ functions with and without moderation. In the moderation case, the variance was set to 0.5. Zero variance corresponds to no moderation. A reference function that takes unit value is also shown.

Forward Stagewise Additive Model View: We now show using (3) that the SGPC design using Algorithm 1 with (5) as the objective function (to select the basis vectors and their coefficients) is equivalent to building a forward stagewise additive model. In particular, a basis vector selection results in a basis function choice and the coefficient optimization essentially results in its site parameters estimation in each iteration (steps 3 and 4 in Algorithm 1). Note that the notions of stage and iteration in Algorithm 1 are equivalent. First, let us look at the steps 3 and 4 of Algorithm 1 more closely. After selecting a basis vector j and updating its site parameters (p_j, m_j) at the t -th iteration, the following posterior variance and mean update can be obtained by simplifying (3):

$$\text{diag}(\mathbf{A})^{(t+1)} := \text{diag}(\mathbf{A})^{(t)} - \eta_j \tilde{\mathbf{k}}_{.,j}^2, \quad \hat{\mathbf{f}}^{(t+1)} := \hat{\mathbf{f}}^{(t)} + \tilde{\alpha}_j \tilde{\mathbf{k}}_{.,j} \quad (7)$$

where $\tilde{\mathbf{k}}_{.,j} = (\mathbf{k}_{.,j} - \mathbf{k}_{.,\mathbf{u}_t} \mathbf{\Pi}_{\mathbf{u}_t}^{\frac{1}{2}} \mathbf{B}_{\mathbf{u}_t}^{-1} \mathbf{\Pi}_{\mathbf{u}_t}^{\frac{1}{2}} \mathbf{k}_{\mathbf{u}_t, .})$ and \mathbf{u}_t is the basis vector set at the t -th iteration. Here $\eta_j = \frac{p_j}{1 + p_j \mathbf{A}_{jj}^{(t)}}$ and $\tilde{\alpha}_j = \eta_j (m_j - \hat{f}_j^{(t)})$. Note that $\eta_j \geq 0$. Then the process of adding the j th basis vector is equivalent to adding a basis function $\tilde{\mathbf{k}}(\mathbf{x}, \mathbf{x}_j)$. That is, we can define the additive model function for SGPC as: $\hat{f}^{(t+1)}(\mathbf{x}) = \hat{f}^{(t)}(\mathbf{x}) + \tilde{\alpha}_j \tilde{k}(\mathbf{x}, \mathbf{x}_j)$. Here, $\tilde{k}(\mathbf{x}, \mathbf{x}_j) = k(\mathbf{x}, \mathbf{x}_j) - \mathbf{k}(\mathbf{x}, \mathbf{x}_{\mathbf{u}_t}) \mathbf{\Pi}_{\mathbf{u}_t}^{\frac{1}{2}} \mathbf{B}_{\mathbf{u}_t}^{-1} \mathbf{\Pi}_{\mathbf{u}_t}^{\frac{1}{2}} \mathbf{k}(\mathbf{x}_{\mathbf{u}_t}, \mathbf{x})$ (where $\mathbf{k}(\mathbf{x}, \mathbf{x}_{\mathbf{u}_t})$ is a row vector of size $|\mathbf{u}_t|$), and

is dependent on the input \mathbf{x}_j through $k(\mathbf{x}, \mathbf{x}_j)$, the previously chosen functions and their site parameters. Note that in both the ADF approximation and the proposed methods, the site parameters of the previously selected basis vectors are not updated whenever a new basis vector is added. This is done to reduce the computational complexity. Next, we can see that the choice of $\tilde{\alpha}_j$ is dependent on the site parameters m_j and p_j . This is because $\hat{f}_j^{(t)}$ and $\mathbf{A}_{jj}^{(t)}$ are fixed once the j th basis vector is chosen. Now, relating $\hat{f}^{(t+1)}(\mathbf{x})$ to the predictive mean vector in (7), we see that the predictive mean vector is nothing but the evaluation of the function $\hat{f}^{(t+1)}(\mathbf{x})$ for the training inputs $\mathbf{x}_i, i = 1, \dots, n$. Therefore, selection of the j th basis vector and estimation of its site parameters (p_j, m_j) in each iteration (stage) of the SGPC design algorithm essentially determine the basis function $\tilde{k}(\mathbf{x}, \mathbf{x}_j)$ and its coefficient $\tilde{\alpha}_j$. To summarize, we have the final classifier function (excluding the bias hyperparameter b) and the predictive variance on an input \mathbf{x} as:

$$\hat{f}(\mathbf{x}) = \sum_{i=1}^{d_{max}} \tilde{\alpha}_i \tilde{k}(\mathbf{x}, \mathbf{x}_i) \quad (8)$$

$$\hat{\sigma}^2(\mathbf{x}) = k(\mathbf{x}, \mathbf{x}) - \sum_{i=1}^{d_{max}} \eta_i \tilde{k}(\mathbf{x}, \mathbf{x}_i) \quad (9)$$

Note that the expression for $\hat{\sigma}^2(\mathbf{x})$ follows from the expression for $\text{diag}(\mathbf{A})^{(t+1)}$ on the left hand side of (7). It is interesting to see that the variance is a non-increasing function as more and more basis functions are added. Having shown the equivalence, we next show how the j th basis function and the associated coefficient $\tilde{\alpha}_j$ can be obtained by optimizing (5) in each iteration. As we have seen before, the choice of a basis vector determines the basis function and we describe next how this selection is done.

Basis Vector Selection Method: From efficiency viewpoint, we propose to select a basis vector as:

$$j = \arg \min_{i \in \mathbf{J}} \text{NLP}_a(\{\mathbf{u} \cup i\}, \boldsymbol{\theta}). \quad (10)$$

where \mathbf{J} , a working set, is a randomly chosen subset of \mathbf{u}^c , $|\mathbf{J}| = \min(\kappa, |\mathbf{u}^c|)$ and κ can be set to 59 [10]. To select one basis vector using (10) the computational cost is $O(\kappa n d_{max})$. Therefore a method to reduce the factor κ without significantly degrading generalization performance will be very useful. We achieve this by changing the sampling strategy (from random sampling) used to construct the working set \mathbf{J} . In the proposed adaptive sampling technique, we construct \mathbf{J} by sampling from \mathbf{u}^c according to a distribution that changes after a basis vector is added in each iteration. The sampling distribution is given by:

$$\chi_{j \in \mathbf{u}_t^c}^{(t+1)} = \frac{1}{V^{(t)}} \left(1 - \Phi \left(\frac{y_j(\hat{f}_j^{(t)} + b)}{\sqrt{1 + \mathbf{A}_{jj}^{(t)}}} \right) \right) \quad (11)$$

where $V^{(t)}$ is a normalizing constant. Here, $\hat{\mathbf{f}}_j^{(t)}$ and $\mathbf{A}_{jj}^{(t)}$ are computed using the basis vectors in \mathbf{u}_t . Since $\hat{\mathbf{f}}$ and \mathbf{A} change after inclusion of every basis vector in the inner loop, the distribution also changes and the sampling becomes adaptive.

To understand why such a sampling along with (10) would be useful, we can see that if $\Phi(\cdot) \rightarrow 1$ (for a correctly classified example with high predictive probability), then the probability of selecting such an example as a basis vector will be relatively small. On the other hand, the probability of selecting a misclassified example with low predictive probability (that is, $\Phi(\cdot) \rightarrow 0$) will be relatively high. We found that selecting the most violated example (that is, the example with the least $\Phi(\cdot)$ in \mathbf{u}^c) in each iteration results in poor basis vector selection for noisy and difficult datasets. The adaptive sampling technique can safeguard against such a selection and is robust across different datasets. Next, the sign of $\tilde{\alpha}_j$ in (7) gets adjusted in such a way that $\hat{\mathbf{f}}^{(t+1)}$ moves in the desired direction for a given $\tilde{\mathbf{k}}_{.,j}$. This desired movement is expected to happen for all the examples having same class label that are close enough to the j th example. Therefore, with a choice of an example (having low value of $\Phi(\cdot)$), $\hat{\mathbf{f}}^{(t+1)}$ moving in the desired direction and variance $\text{diag}(\mathbf{A})^{(t+1)}$ non-increasing, we expect the NLP value in (5) to improve particularly for the examples with wrong predictions or low predictive probability. In this sense the basis vector selection using (10) and (11) tends to mimic the selection of a base classifier in boosting [7] that minimizes the training set error with weighted distribution. This helps in getting a better generalization performance for a fixed κ compared to random sampling. Alternatively, κ can be reduced to get the same generalization performance. Experimental results support these claims.

Site Parameters Optimization Method: Having constructed the working set \mathbf{J} using the adaptive sampling technique, we optimize (5) to find $\tilde{\alpha}_i$ for each basis vector $i \in \mathbf{J}$. As shown earlier optimizing over $\tilde{\alpha}_i$ is equivalent to optimizing over the site parameters m_i and p_i for a given basis vector. Essentially we have a two dimensional (m_i, p_i) non-linear optimization problem. Note that it is a constrained optimization problem (under certain condition given below) since the posterior variance $\text{diag}(\mathbf{A})^{(t+1)}$ should be non-negative after every iteration. Assuming that $\text{diag}(\mathbf{A})^{(t)}$ is non-negative it turns out that p_i must satisfy: $\tilde{\eta}_i \geq \eta_i = \frac{p_i}{1+p_i \mathbf{A}_{ii}^{(t)}}$ where $\tilde{\eta}_i = \min_l \{ \frac{\mathbf{A}_{li}^{(t)}}{k_{l,i}^2} \}$. On further simplification we find that if $\tilde{\eta}_i \mathbf{A}_{ii}^{(t)} \geq 1$ then we have an unconstrained optimization problem (in τ_i when we work with $p_i = \exp(\tau_i)$); otherwise we have a constrained optimization problem with $0 \leq p_i \leq \frac{\tilde{\eta}_i}{1-\tilde{\eta}_i \mathbf{A}_{ii}^{(t)}}$. This can be solved using any standard nonlinear optimization technique. *To summarize, we construct \mathbf{J} using adaptive sampling, optimize $\tilde{\alpha}_i$, $\forall i \in \mathbf{J}$ and select the basis vector using (10).*

4 Related work

In this section we briefly describe three closely related methods that we compare with the proposed method. In entropy based method [4], a basis vector is chosen

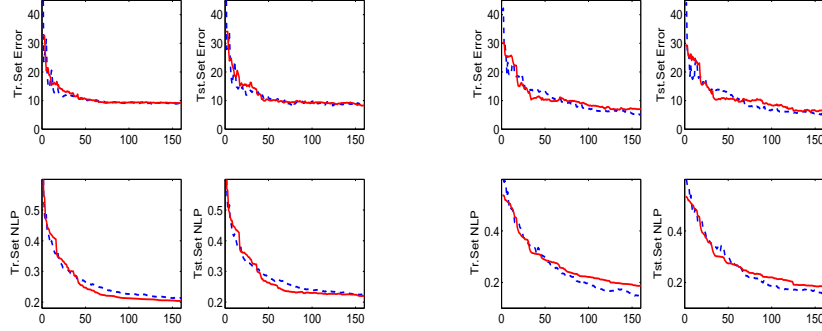


Fig. 2. The left panel (group of 4 plots) corresponds to Waveform dataset and the right panel corresponds to Image dataset. The first and second rows show the training/test set errors and NLP loss as the basis vectors are added in the inner loop just before termination. The solid-red and dashed-blue lines correspond to $\tilde{\alpha}$ with moment matching and constrained optimization cases with ADF approximation. In this experiment we set $\kappa = 2$.

according to the change in the entropy of the posterior process (1) after inclusion in the model and is given by: $j = \arg \min_{i \in \mathbf{u}^c} \log(\bar{\lambda}_i)$ where $\bar{\lambda}_i = 1 - \nu_i \mathbf{A}_{ii}$. In information gain based method [8], a basis vector is chosen according to the information gain (which is defined as negative of the Kullback-Leibler divergence) obtained from the posterior process after inclusion in the model and is given by: $j = \arg \min_{i \in \mathbf{u}^c} \left\{ -\log(\bar{\lambda}_i) + \frac{1}{\bar{\lambda}_i} + \frac{(\hat{f}_i' - \hat{f}_i)^2}{\mathbf{A}_{ii}} \right\}$. Here \hat{f}_i and \hat{f}_i' denote the predictive mean before and after the inclusion of the i th basis vector. Compared to the entropy based selection, this method takes the predictive mean also into account and differs from the way $\bar{\lambda}_i$ is traded-off between the first and second term. Both these methods are very efficient since the relevant quantities that are needed to compute the appropriate measure for the basis vector selection are maintained throughout in the inner loop of Algorithm 1. Both these methods maximize the marginal likelihood for hyperparameter optimization.

In validation based method [9], a working set $\mathbf{J} \subseteq \mathbf{u}^c$ of fixed size κ ($\kappa = \min(|\mathbf{u}^c|, 59)$) is constructed by sampling randomly from \mathbf{u}^c . The basis vector that minimizes (4) is chosen and this involves computation of a new NLP value after inclusion for each $i \in \mathbf{J}$. Thus the computational cost for one basis vector selection is $O(\kappa nd_{max})$. The hyperparameters are selected by minimizing (4).

While all the three methods use moment matching with the ADF approximation to estimate the site parameters (2), the proposed site parameters optimization method provides an alternate way to estimate these parameters. Note that one can also use (2) in conjunction with the proposed adaptive sampling based basis vector selection method. On comparing the objective functions used by the proposed method and validation based method, we see that the form of (4) is same that of (5) except that the summation happens only over \mathbf{u}^c . While the validation based method viewed (4) as obtaining the NLP performance estimate with a validation set, (5) is motivated from the additive modeling viewpoint and, minimizing an upper bound on the training set error. Furthermore, the

validation based method uses fixed uniform sampling instead of adaptive sampling. Note that the difference between (4) and (5) is expected to be insignificant when $d_{max} \ll n$ (usually the case in SGPC design with large datasets) and this condition is important to avoid any overfitting.

5 Experiments

The summary of the datasets used in the experiments is given in Table 1. These datasets are part of Gunnar Raetsch’s benchmark datasets available at <http://theoval.cmp.uea.ac.uk/~gcc/matlab/default.html>. We changed the training and test set sizes of the top five datasets in Table 1 to demonstrate the effectiveness of the proposed method on large datasets. For the first four datasets we picked top 3600 test examples from the original test set partition and added to the training set. The remaining examples were used as the test set. Note that this construction however results in reduction of the test set size. In the case of *Splice* dataset, we picked the top 1000 examples from the test set partition. The modified train and test set sizes are shown Table 1. We considered only the first 25 partitions of the first four datasets. In all the experiments we used the squared exponential covariance function and Algorithm 1 described in Section 2. A conjugate gradient method was used to optimize (4) (unless otherwise specified) in the outer loop for optimizing the hyperparameters, and $iter_{max}$ was set to 20. We kept track of the best model based on the NLP loss value after every outer loop iteration. For comparison, we evaluated the test set error and NLP loss performance.

Table 1. Datasets Description. n and m denote the training and test set sizes. d and pt denote the input dimension and number of partitions.

Dataset	n	m	d	pt
Banana	4000	1300	2	25
Waveform	4000	1000	21	25
Twonorm	4000	3400	20	25
Ringnorm	4000	3400	20	25
Splice	2000	1175	60	20
Image	1300	1010	18	20

We conducted three experiments. Due to the space constraints we present only selected results. In the first experiment we illustrate the effectiveness of the proposed method of site parameters (equivalently, $\tilde{\alpha}$) optimization. The results on one partition of the **Waveform** and **Image** datasets are shown in Figure 2. This method is compared against using (2) for site parameters optimization. Although some minor variations were seen between the two methods, statistical analysis showed that the performance differences were not significant. Thus, the constrained optimization is an effective alternate method to estimate the site parameters. We now discuss certain practical aspects of this optimization. During optimization, the variance can become zero (within numerical accuracy), for some choice of the hyperparameter values and, also due to the greedy nature of the basis vector selection method. While this can be handled in some way

(for example by exiting the inner loop), optimizing over individual $\tilde{\alpha}_i$'s can become slightly expensive for large datasets. Note that the function and gradient computations are linear in n . We can control the optimization cost by restricting the number of function and gradient evaluations with some inaccuracy in the solution. Therefore, the proposed optimization is also efficient.

In the next two experiments, we kept the site parameter estimation (using (2)) and the hyperparameter estimation (using (4)) same, and only changed the basis vector selection method in the step 3 of Algorithm 1. This is because our goal here is to compare the quality of the different basis vector selection methods. First, we demonstrate the effectiveness of the adaptive sampling method in the basis vector selection. This is done by comparing it with random (uniform) sampling method. We conducted this experiment on all the datasets given in Table 1. The test set error and NLP loss performance results on two datasets are given in Figure 3 (left panel) for two different values of d_{max} . These results were obtained by averaging the performance over the partitions. We found that the adaptive sampling method consistently performed better across all the datasets, particularly with respect to the NLP loss measure. This is because the choice of the basis vectors made by the adaptive sampling method is based on the predictive distribution. We also observed improved test set error performance on several cases. It was also observed that the performance difference reduces as the working set size κ increases. It can also be seen that κ value of 2 is sufficient for the adaptive sampling method to get similar NLP generalization performance as the validation based method (see the second column in the left panel of Figure 3).

In the third experiment, we compared the performance of the proposed method, validation based method, entropy and information gain based basis vector selection methods. In the case of proposed method, we evaluated the performance with $\kappa = 1$ and 2, thus ensuring that the complexity for the basis vector selection is the same as that of the entropy and information gain based methods. We conducted this experiment for four different values of d_{max} (40, 80, 160 and 320) on all the datasets given in Table 1. The test set error and NLP loss performance on three datasets are shown in Figure 3 (right panel). They were obtained by averaging the performance over the partitions. We compared the performance of various methods using statistical significance tests. We first conducted Wilcoxon test on the test set error and NLP loss obtained from the partitions, on each dataset. All the observations from the tests below are made at the significance level of 0.05. The results indicated better test set error and NLP performance of the proposed method over the entropy and information gain based methods on almost all the datasets. Specifically we observed that the proposed method performed better on difficult datasets (relatively higher test set errors) like **Banana**, **Waveform** and **Splice** for all values of d_{max} with respect to (w.r.t.) both the measures. On **Twonorm** and **Ringnorm** datasets it performed better w.r.t. the NLP loss measure for all the values of d_{max} . While it performed better than the entropy based method on the **Ringnorm** dataset for all values of d_{max} w.r.t. the test set error, the performance was the same at higher values of

d_{max} in other cases. The information gain based method performed better than the entropy based method on the **Banana**, **Waveform** and **Ringnorm** datasets. The entropy based method performed better than the information gain based method w.r.t. the test set error in the case of **Twonorm** dataset. We observed that the entropy based method performed better than all the methods at lower values of d_{max} on the **Image** dataset. On comparing the proposed method ($\kappa = 1$ and 2) with the validation based method, we found that the validation based method performed better w.r.t. the test set error at lower values of d_{max} (40 and 80).

Next, following [2], we conducted Friedman’s test with six datasets (Table 1) and four methods, namely, the proposed method (with $\kappa=2$), validation, entropy and information gain based methods. To conduct this test, we used the average test set error and NLP values obtained from averaging over the partitions. The p-values obtained for the test set error and NLP measure were (0.02, 0.04, 0.04, 0.39) and (0.002, 0.002, 0.01, 0.09) respectively for four different values of d_{max} (40, 80, 160 and 320) in that order. When d_{max} was 320, the results were not significantly different w.r.t. both the measures. Since the null hypothesis was rejected for d_{max} values of 40, 80 and 160, we next conducted the Bonferroni-Dunn post-hoc test to compare the proposed method with the other three methods. This test revealed that there were no significant differences between the proposed and validation based methods for all values of d_{max} w.r.t. both the measures. On comparing the proposed method with the entropy and information gain based methods, we found that while the results were not significantly different w.r.t. the test set error, they were significant w.r.t. the NLP measure for lower d_{max} values at 0.1 level. Overall, it was seen that the p-value became larger and the performance differences across the methods reduced as d_{max} was increased.

Except for the validation based method ($\kappa = 59$), all the methods required almost the same computational time for the basis vector selection. An approximate timing measurement of one inner loop (for $d_{max}=80$) showed that the proposed method with $\kappa = 1$ took approximately 20 seconds for the **Banana** dataset (on a machine with 2 GB of RAM and dual core Intel CPU running at 1.83 GHz). In general, we found that the proposed method was 5 times faster than the validation based method on almost all the datasets. This comparison was based on the Matlab implementations of these methods. The speed improvement was not as high as 59. We believe that efficient matrix based operations in Matlab helped the validation based method significantly and, expect the speed improvement to be higher with implementations in other programming languages like C.

6 Conclusion

We considered the problem of designing an SGPC from an additive model estimator viewpoint. We introduced new methods for basis vector selection and site parameters estimation based on the predictive loss function. An adaptive sampling method that aids in effective basis vector selection and computational complexity reduction was proposed. The proposed basis vector selection method has same computational and storage complexities as that used by IVM and, is

thus suitable for large datasets. The experimental results showed better generalization performance of the proposed method on several benchmark datasets, particularly for relatively smaller d_{max} values or on difficult datasets.

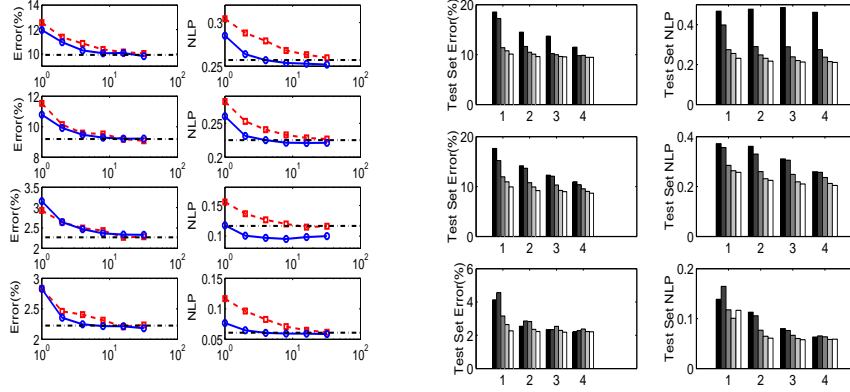


Fig. 3. Left Panel of eight plots: Test set error and NLP loss performance of the random sampling (dashed-red-square) and adaptive sampling (solid-blue-circle) methods for different values of κ . The dashed-dot-black line corresponds to the validation based method with $\kappa=59$. Top two rows correspond to Waveform dataset for $d_{max}=40$ and 80 (in that order). The bottom rows correspond to Twonorm dataset for $d_{max}=40$ and 80. **Right Panel of six plots:** Test set performance of the various basis vector selection methods (entropy, information-gain, proposed method with $\kappa=1$ and 2, and validation based method ($\kappa=59$)) (different gray shades) in that order) for different values of d_{max} (40, 80, 160 and 320 correspond to the x-axis values of 1, 2, 3 and 4 respectively). Each row corresponds to one dataset. The results on Banana, Waveform and Twonorm datasets are given in that order.

References

1. L. Csató. *Gaussian processes - iterative sparse approximation*. PhD thesis, Aston University, Birmingham, UK, 2002.
2. J. Demsar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.
3. J. H. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting (with discussion). *Annals of Statistics*, 28:337–407, 2000.
4. N. D. Lawrence, M. Seeger, and R. Herbrich. Fast sparse Gaussian process methods: The informative vector machine. In *Advances in Neural Information Processing Systems*, 2003.
5. T. P. Minka. *A family of algorithms for approximate Bayesian inference*. PhD thesis, Massachusetts Institute of Technology, 2001.
6. C. E. Rasmussen and C. K. I. Williams. *Gaussian processes for machine learning*. MIT Press, 2006.

7. R. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. In *Proc. of the Eleventh Annual Conference on COLT*, 1998.
8. M. Seeger, N. D. Lawrence, and R. Herbrich. Efficient nonparametric Bayesian modelling with sparse Gaussian process approximations. Technical report, <http://www.kyb.tuebingen.mpg.de/bs/people/seeger>, 2007.
9. S. Shevade and S. Sundararajan. Validation-based sparse Gaussian process classifier design. *Neural Computation*, 21(7):2082–2103, 2009.
10. A. J. Smola and P. Bartlett. Sparse greedy Gaussian process regression. In *Advances in Neural Information Processing Systems*, 2001.